HOMEWORK #1 - SQL

OVERVIEW

The first homework is to construct a set of SQL queries for analysing a dataset that will be provided to you. For this, you will look into IMDB data. This homework is an opportunity to:
(1) learn basic and certain advanced SQL features, and (2) get familiar with using a full-featured DBMS, SQLite, that can be useful for you in the future.
This is a single-person project that will be completed individually (i.e., no groups).

- Release Date: Sep 01, 2019
- Due Date: Sep 11, 2019 @ 11:59pm

SPECIFICATION

The homework contains 10 questions in total and is graded out of 100 points. For each question, you will need to construct a SQL query that fetches the desired data from the SQLite DBMS. It will likely take you approximately 6-8 hours to complete the questions.

PLACEHOLDER FOLDER

Create the placeholder submission folder with the empty SQL files that you will use for each question:

```
$ mkdir placeholder
$ cd placeholder
$ touch q1_sample.sql \
        q2_uncommon_type.sql \
        q3_tv_vs_movie.sql \
        q4_old_is_not_gold.sql \
        q5_percentage.sql \
        q6_dubbed_smash.sql \
        q7_imdb_250.sql \
        q8_number_of_actors.sql \
        q9_movie_names.sql \
        q10_genre_counts.sql
```

After filling in the queries, you can compress the folder by running the following command:

```
$ zip -j submission.zip placeholder/*.sql
```

The `-j` flag lets you compress all the SQL queries in the zip file without path information. The grading scripts will not work correctly unless you do this.

INSTRUCTIONS

SETTING UP SQLITE

You will first need to install SQLite on your development machine.

INSTALL SQLITE3 ON UBUNTU LINUX

Install the `sqlite3` and `libsqlite3-dev` packages by running the following command;

```
$ sudo apt-get install sqlite3 libsqlite3-dev
```

INSTALL SQLITE3 ON MAC OS X

On Mac OS Leopard or later, you don't have to! It comes pre-installed. You can upgrade it, if you absolutely need to, with Homebrew.

LOAD THE DATABASE DUMP

1. Check if `sqlite3` is properly working by following this tutorial.
2. Download the database dump file:

```
3. $ wget https://15445.courses.cs.cmu.edu/fall2019/files/imdb-cmudb2019.db.gz
```

Check its MD5 checksum to ensure that you have correctly downloaded the file:

```
$ md5 imdb-cmudb2019.db.gz
MD5 (imdb-cmudb2019.db.gz) = 6443351d4b55eb3c881622bd60a8dc5b
```

1. Unzip the database from the provided database dump by running the following commands on your shell. Note that the database file be 900MB after you decompress it.

```
$ gunzip imdb-cmudb2019.db.gz
$ sqlite3 imdb-cmudb2019.db
```

We have prepared a random sample of the original dataset for this assignment. Although this is not required to complete the assignment, the complete dataset is available by following the steps here.

1. Check the contents of the database by running the `.tables` command on the `sqlite3` terminal. You should see 6 tables, and the output should look like this:

```
$ sqlite3 imdb-cmudb2019.db
SQLite version 3.11.0
Enter ".help" for usage hints.
sqlite> .tables
akas       crew       episodes   people     ratings    titles
```

1. Create indices using the following commands in SQLite:

```
CREATE INDEX ix_people_name ON people (name);
CREATE INDEX ix_titles_type ON titles (type);
CREATE INDEX ix_titles_primary_title ON titles (primary_title);
CREATE INDEX ix_titles_original_title ON titles (original_title);
CREATE INDEX ix_akas_title_id ON akas (title_id);
CREATE INDEX ix_akas_title ON akas (title);
CREATE INDEX ix_crew_title_id ON crew (title_id);
CREATE INDEX ix_crew_person_id ON crew (person_id);
```

CHECK THE SCHEMA

Get familiar with the schema (structure) of the tables (what attributes do they contain, what are the primary and foreign keys). Run the `.schema $TABLE_NAME` command on the `sqlite3` terminal for each table. The output should look like the example below for each table.

PEOPLE

```
sqlite> .schema people
CREATE TABLE people (
  person_id VARCHAR PRIMARY KEY,
  name VARCHAR,
  born INTEGER,
  died INTEGER
);
CREATE INDEX ix_people_name ON people (name);
```

Contains details for a person. For example, this is a row from the table:

```
nm0000003|Brigitte Bardot|1934|
```

For us, the important fields are `person_id` (e.g., "nm0000003"), `name` (e.g., "Brigitte Bardot"), and the year that the person was `born` (e.g., "1934").

TITLES

```
sqlite> .schema titles
CREATE TABLE titles (
  title_id VARCHAR PRIMARY KEY,
  type VARCHAR,
  primary_title VARCHAR,
  original_title VARCHAR,
  is_adult INTEGER,
  premiered INTEGER,
  ended INTEGER,
  runtime_minutes INTEGER,
  genres VARCHAR
);
CREATE INDEX ix_titles_type ON titles (type);
CREATE INDEX ix_titles_primary_title ON titles (primary_title);
CREATE INDEX ix_titles_original_title ON titles (original_title);
```

Contains details for a title. For example, this is a row from the table:

```
tt0081400|movie|Raise the Titanic|Raise the Titanic|0|1980||115|Action,Adventure,D
rama
```

For us, the important fields are `title_id` (e.g., "tt0081400"), `type` (e.g., "movie"), `primary_title` (e.g., "Raise the Titanic"), `runtime_minutes` (e.g., 127) and `genres` (e.g., "Action,Adventure,Drama").

AKAS

```
CREATE TABLE akas (
  title_id VARCHAR, -- REFERENCES titles (title_id),
  title VARCHAR,
  region VARCHAR,
  language VARCHAR,
  types VARCHAR,
  attributes VARCHAR,
  is_original_title INTEGER
);
CREATE INDEX ix_akas_title_id ON akas (title_id);
CREATE INDEX ix_akas_title ON akas (title);
```

The table contains the alternate titles for the dubbed movies. Note that `title_id` in this table corresponds to `title_id` in `titles`.

CREW

```
CREATE TABLE crew (
  title_id VARCHAR, -- REFERENCES titles (title_id),
  person_id VARCHAR, -- REFERENCES people (person_id),
  category VARCHAR,
  job VARCHAR
);
CREATE INDEX ix_crew_title_id ON crew (title_id);
CREATE INDEX ix_crew_person_id ON crew (person_id);
```

Contains the details of the cast of the title. For example, this is a row from the table:

```
tt0000003|nm5442194|producer|producer
```

For us, the important fields are `title_id` (e.g., "tt0000003"), `person_id` (e.g., "nm5442194"), and `category` (e.g., "producer").
Note that `title_id` corresponds to `title_id` in `titles` and `person_id` corresponds to `person_id` in `people`.

RATINGS

```
CREATE TABLE ratings (
```

```
  title_id VARCHAR PRIMARY KEY, -- REFERENCES titles (title_id),
  rating FLOAT,
  votes INTEGER
);
```

Contains the ratings for each title. For example, this is a row from the table:

```
tt0000001|5.6|1529
```

For us, the important fields are `title_id` (e.g., "tt0000001"), `rating` (the average rating for all votes, e.g., "5.6") and `votes` (the number of votes for the title, e.g., "1529").

SANITY CHECK

Count the number of rows in the table

```
sqlite> select count(*) from titles;
2294719
```

CONSTRUCT THE SQL QUERIES

Now, it's time to start constructing the SQL queries and put them into the placeholder files.

Q1 [0 POINTS] (Q1_SAMPLE):

The purpose of this query is to make sure that the formatting of your output matches exactly the formatting of our auto-grading script.
Details: List all distinct types of titles ordered by type.
Answer: Here's the correct SQL query and expected output:

```
sqlite> select distinct(type) from titles order by type;
    movie
    short
    tvEpisode
    tvMiniSeries
    tvMovie
    tvSeries
    tvShort
    tvSpecial
    video
    videoGame
```

You should put this SQL query into the appropriate file (`q1_sample.sql`) in the submission directory (`placeholder`).

Q2 [5 POINTS] (Q2_UNCOMMON_TYPE):

List the longest title of each type along with the runtime minutes.

length(str)

Details: Find the titles which are the longest by runtime minutes. There might be cases where there is a tie for the longest titles - in that case return all of them. Display the types, primary titles and runtime minutes, and order it according to type (ascending) and use primary titles (ascending) as tie-breaker.

Q3 [5 POINTS] (Q3_TV_VS_MOVIE):

List all types of titles along with the number of associated titles.
Details: Print type and number of associated titles. For example, `tvShort|4075`. Sort by number of titles in ascending order.

Q4 [10 POINTS] (Q4_OLD_IS_NOT_GOLD):

Which decades saw the most number of titles getting premiered? List the number of titles in every decade. Like `2010s|2789741`.
Details: Print all decades and the number of titles. Print the relevant decade in a fancier format by constructing a string that looks like this: `2010s`. Sort the decades in decreasing order with respect to the number of titles. Remember to exclude titles which have not been premiered (i.e. where `premiered` is `NULL`)
PS: Add this to your watchlist: 100 Years (2115)

CAST(… AS TEXT/ int / real)

Q5 [10 POINTS] (Q5_PERCENTAGE):

List the decades and the percentage of titles which premiered in the corresponding decade. Display like : `2010s|45.7042`
Details: The percentage of titles for a decade is the number of titles which premiered that decade divided by the total number of titles. For the total number of titles, count all titles including ones that have not been premiered. Round the percentage to four decimal places using `ROUND()`.

Q6 [10 POINTS] (Q6_DUBBED_SMASH):

List the top 10 dubbed titles with the number of dubs.
Details: Count the number of titles in `akas` for each title in the `titles` table, and list only the top ten. Print the primary title and the number of corresponding dubbed movies.

limit 10

Q7 [15 POINTS] (Q7_IMDB_250):

List the IMDB Top 250 movies along with its weighted rating.
Details: The weighted rating of a movie is calculated according to the following formula:

```
Weighted rating (WR) = (v/(v+m)) * R + (m/(v+m)) * C
```

- - R = average rating for the movie (mean), i.e. ratings.rating
- - v = number of votes for the movie, i.e. ratings.votes
- - m = minimum votes required to be listed in the Top 250 (current 25000)
- - C = weighted average rating of all movies

Print the movie name along with its weighted rating. For example: `The Shawshank Redemption|9.27408375213064`.

Q8 [15 POINTS] (Q8_NUMBER_OF_ACTORS):

List the number of actors / actresses who have appeared in any title with Mark Hamill (born in 1951).
Details: Print only the total number of actors and actresses. The answer should include Mark Hamill himself.

Q9 [15 POINTS] (Q9_MOVIE_NAMES):

List the movies in alphabetical order which cast both Mark Hamill (born in 1951) and George Lucas (born in 1944).
Details: Print only the names of the movies in alphabetical order.

Q10 [15 POINTS] (Q10_GENRE_COUNTS):

List all distinct genres and the number of titles associated with them.
Details: The `titles` table contains the titles with their genres. Each title is associated with zero or more genres stored in the `genres` column as comma-separated values (like "Documentary,Short"). Count the number of titles associated with each genre, and list the genres and the counts, and order it according to the counts (greatest to least). Don't forget to filter empty genres (where `genres` is blank).
Hint: You might find CTEs useful.

GRADING RUBRIC

Each submission will be graded based on whether the SQL queries fetch the expected sets of tuples from the database. Note that your SQL queries will be auto-graded by comparing their outputs (i.e. tuple sets) to the correct outputs. For your queries, the order of the output columns is important; their names are not.

LATE POLICY

See the late policy in the syllabus.

SUBMISSION

You can submit on Gradescope now, but we are currently setting up the autograder. We will post on Piazza when the autograder is up.
We use the Autograder from Gradescope for grading in order to provide you with immediate feedback. After completing the homework, you can submit your compressed folder `submission.zip` (only one file) to Gradescope:

- https://www.gradescope.com/courses/58985/

Important: Use the Gradescope course code announced on Piazza.
We will be comparing the output files using a function similar to `diff`. You can submit your answers as many times as you like.

COLLABORATION POLICY

- Every student has to work individually on this assignment.
- Students are allowed to discuss high-level details about the project with others.
- Students are not allowed to copy the contents of a white-board after a group meeting with other students.
- Students are not allowed to copy the solutions from another colleague.

WARNING: All of the code for this project must be your own. You may not copy source code from other students or other sources that you find on the web. Plagiarism will not be tolerated. See CMU's Policy on Academic Integrity for additional information.

Last Updated: Sep 06, 2019